padnon : un logiciel de tracé automatique de graphes en trois dimensions

François Bertault

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Padnon : *un logiciel de tracé automatique de graphes en trois dimensions*

François Bertault

N■ 3130

Mars 1997

THÈME 2

ISSN 0249-6399

INRIA
LORRAINE

Padnon : un logiciel de tracé automatique de graphes en trois dimensions

François Bertault

Thème 2 — Génie logiciel
et calcul symbolique
Projet Euréca

Rapport de recherche n° 3130 — Mars 1997 — 17 pages

Résumé : La représentation graphique des graphes est un problème dont les applications pratiques sont nombreuses. Nous présentons dans cet article un logiciel de tracé automatique de graphes orientés ou non-orientés, appelé `Padnon`, permettant de les visualiser et de les modifier dans un espace en trois dimensions. Ce programme peut être relié à un autre programme, pour servir d'interface graphique de représentation dynamique de graphes. L'algorithme de tracé est conçu pour ne modifier que légèrement le tracé du graphe lors de modifications incrémentales et décrémentales de ce dernier. `Padnon` est, pour cette raison, particulièrement bien adapté aux problèmes d'animations d'algorithmes utilisant des structures de graphe. Après une présentation du programme, l'algorithme de tracé est décrit en détail.

Mots-clé : Tracé de graphes, algorithme dynamique, interface graphique, animation d'algorithmes

(Abstract: pto)

Unité de recherche INRIA Lorraine
Technopôle de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY (France)
Téléphone : (33) 83 59 30 30 – Télécopie : (33) 83 27 83 19
Antenne de Metz, technopôle de Metz 2000, 4 rue Marconi, 55070 METZ
Téléphone : (33) 87 20 35 00 – Télécopie : (33) 87 76 39 77

Automatic drawing of graphs in three dimensions with Padnon

Abstract: We present in the report a new three dimension graph drawing system, called `Padnon`. `Padnon` deals with both directed and undirected graphs. It allows dynamic modifications of a graph, specified from an other program. Small modifications in the specification induce only small modifications of the representation of the graph. The program is well suited for animating algorithms that use graph data structures. After a presentation of `Padnon`, we describe in detail the layout algorithm.

Key-words: Graph drawing, dynamic algorithm, graphical interface, algorithm animation

Introduction

Les graphes sont utilisés dans de nombreuses applications, pour représenter des relations entre objets. Il sont utilisés dans des domaines très variés, en ingénierie, pour la création de circuits intégrés, en chimie ou encore en sciences sociales par exemple. Quand la taille de ces graphes devient importante, il devient indispensable de pouvoir les représenter graphiquement de façon automatique. Pour cela, on choisit certains critères esthétiques (symétrie, longueur des arêtes par exemple) que l'on souhaite obtenir sur le dessin du graphe, afin de le rendre facilement intelligible. Ces critères sont en général contradictoires, et conduisent souvent à des problèmes difficiles.

Prenons par exemple le cas de la représentation dans le plan des graphes orientés. Une représentation possible consiste à placer les nœuds par niveaux, comme sur la figure 1. Dans le cas où le graphe contient des cycles, on essaye alors de minimiser le nombre d'arêtes qui "remontent".

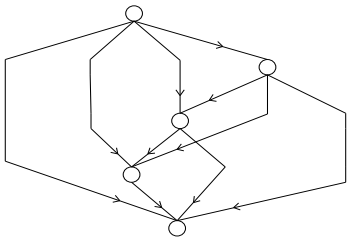


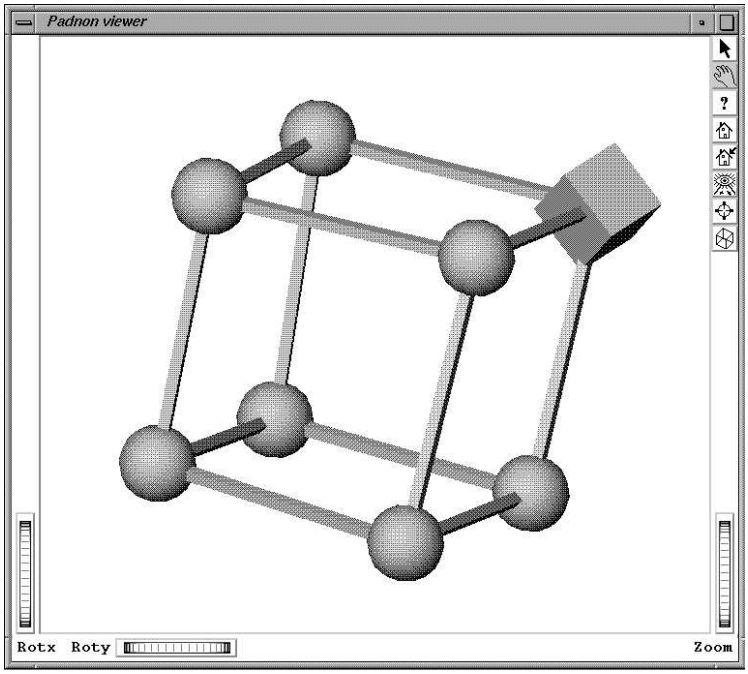
FIG. 1: *Graphe K5 orienté*

Ce premier problème est NP-complet [5]. Le problème suivant, consistant à placer sur les niveaux et à minimiser le nombre de croisements entre les arêtes l'est également [5]. Cet exemple illustre les raisons qui ont conduit à proposer de nombreux algorithmes dédiés à des classes restreintes de graphes ou des critères esthétiques précis, dont on peut évaluer l'étendue dans [2].

Nous présentons dans ce rapport un système mettant en œuvre un algorithme de tracé incrémental, dans le cas de graphes orientés et non orientés. La représentation du graphe se fait dans un espace en trois dimensions, ce qui permet de diminuer les problèmes liés au croisements d'arêtes dans le plan.

1 Présentation du programme `Padnon`

1.1 Fenêtre de visualisation



Fenêtre de visualisation du graphe

L'élément principal du programme `Padnon` est la fenêtre de visualisation du graphe. Cette fenêtre permet de manipuler le graphe de façon intuitive. On peut par exemple, en sélectionnant l'icône *main*, faire tourner le graphe sur lui-même. On peut également se rapprocher ou s'éloigner, se diriger vers un point précis à l'aide de l'icône *cible*, choisir le mode de représentation (fil de fer, faces cachées, textures) à l'aide du menu déroulant accessible avec le bouton droit de la souris. En sélectionnant l'icône *flèche*, on peut faire apparaître, en cliquant sur un nœud, une fenêtre contenant des informations préalablement mémorisées, spécifiques au nœud pointé. Il est également possible de modifier la couleur des nœuds, leur taille, leur position.

Pour charger un graphe d'exemple dans la fenêtre de visualisation (par exemple le graphe décrit dans le fichier `exTest`), il faut, dans la fenêtre de communication, saisir dans le champ `program` la commande `catlayout exTest`, puis appuyer sur le bouton `pipe`.

1.2 Options de tracé

Plusieurs options de tracé sont disponibles pour représenter les graphes à l'aide de `Padnon`. On peut choisir entre trois algorithmes, un pour les graphes orientés (`directed`), l'autre pour les graphes non orientés (`undirected`), et le dernier qui, partant d'une représentation orientée ou non, permet d'afficher la forme générale du graphe, que l'on nomme le *squelette* du graphe (`contract`).

Le nombre d'itérations de l'algorithme de tracé est spécifié par le marqueur `Number of steps`. Le déplacement maximal autorisé des nœuds évolue de façon linéaire au cours des itérations entre les valeurs indiquées par les marqueurs `max. move`. Il est possible de changer la valeur par défaut souhaitée pour la longueur des arêtes, ainsi que la distance maximale au delà de laquelle les nœuds du graphe ne s'écartent plus. La façon dont ces données sont prises en compte est décrite en détail dans la section 2.

Les deux cases `2d` et `node-edge repulsion` permettent d'indiquer si le graphe sera représenté dans un plan et si les nœuds doivent être éloignés des arêtes du graphes. Cette dernière option est en général souhaitable, mais peut-être coûteuse pour des graphes possédant beaucoup d'arêtes.

Les deux dernières cases ne s'appliquent qu'aux graphes orientés, et indiquent comment les nœuds doivent être placés verticalement. La case `incremental` utilise l'algorithme présenté dans la section 2. L'option `Greedy` tend à placer les nœuds qui sont l'origine du plus grand nombre d'arêtes en haut du graphe.

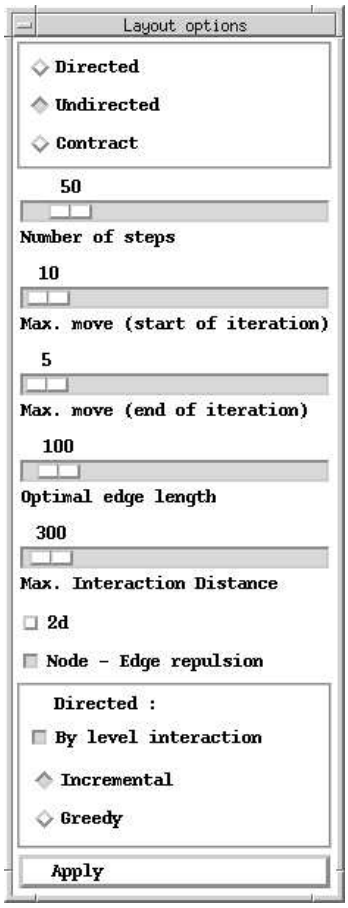


FIG. 2: *Options de tracé*

1.3 Format de description des graphes

Pour qu'une description soit prise en compte par `Padnon`, elle doit être précédé du mot `IMPORT`, placé seul sur une ligne, puis doit se terminer par un point, seul

sur une ligne. Ceci permet de différencier les données qui doivent être interprétées comme étant des graphes, des autres données qui doivent être simplement affichées.

Le format de description des graphes est simple. À chaque nœud du graphe, on associe un entier, appelé l'*identificateur* du nœud. Chaque nœud possède un identificateur différent de ceux des autres nœuds. On procède de la même façon pour les arêtes. Cet identificateur permet d'indiquer simplement les nœuds ou arêtes à modifier ou à détruire. Quand une description de nœud ou d'arête utilise un identificateur déjà utilisé par un nœud du graphe, `Padnon` ne crée pas de nouveau nœud : il modifie le nœud existant avec les nouveaux attributs. La description des nœuds du graphe se fait à l'aide de descriptions de la forme :

```
<nœuds>::=node(id=<entier>,
                shape=<formes>,
                scale=[<entier>,<entier>,<entier>],
                color=[<entier>,<entier>,<entier>],
                x=<entier>,
                y=<entier>,
                z=<entier>,
                info=<texte>,
                label=<texte>,
          )
<formes>::=Sphere|Cube|Cone|Cylinder
```

Dans la description ci-dessus, les champs en gras sont obligatoires. Tous les autres sont facultatifs. Il n'y a pas d'ordre sur les champs. L'indication `<entier>` représente les entiers relatifs, `<texte>` toute chaîne de caractères alpha-numériques, compris entre deux symboles `'`. Le champ `scale`, formé de trois entiers, un pour chaque composante du repère dans lequel est tracé le graphe, permet d'augmenter ou de diminuer la taille d'un nœud. La taille initiale des nœuds correspond à une valeur de 10 pour chaque composante. Par exemple, pour obtenir un nœud trois fois plus haut que les nœuds initiaux, on écrira `scale=[10,10,30]`. Le champ `color` permet d'indiquer la couleur des noeuds. La première valeur du champ indique l'intensité de la composante rouge, la deuxième celle de la verte et la troisième celle de la bleue, avec chaque valeur comprise entre 0 et 255. Les champs `x`, `y` et `z` permettent d'indiquer les coordonnées de certains nœuds. Les valeurs indiquées ne sont pas modifiés par les algorithmes de tracé. Le champ `info` contient les informations qui sont affichées lorsque l'on sélectionne un nœud. Le champ `label` est affiché à proximité du nœud dans la fenêtre de visualisation. Les espaces et retours chariots sont ignorés dans la description, sauf pour les valeurs des champs de type `<texte>`.

La description des arêtes est similaire :

```
<arêtes>::=edge(id=<entier>,
                source=<entier>,
                target=<entier>,
                shape=<formes>,
                scale=[<entier>,<entier>,<entier>],
                color=[<entier>,<entier>,<entier>],
                delta=<entier>,
                arrow=<entier>,
                label=<texte>,
              )
<formes>::=Sphere|Cube|Cone|Cylinder
```

Si le champ `id` n'est pas présent, l'arête est supposée nouvelle. Par contre on perd la possibilité de la modifier ou de la supprimer ultérieurement. Les champs `source` et `target` doivent contenir des identificateurs de nœuds déjà présents dans le graphe. Il est possible de décrire des arêtes réflexives et multiples. Le champ `arrow` vaut 1 si les flèches des arêtes sont dessinées, 0 sinon. Le champ `delta` correspond à la distance verticale souhaitée entre les extrémités de l'arête dans le cas orienté, et à la longueur souhaitée de l'arête dans le cas non orienté.

Il est possible de donner des valeurs par défaut aux différents champs (excepté pour le champ `id`), à l'aide de descriptions identiques à celle présentées ci-dessus, en remplaçant simplement le mot `node` par `nodedefault`, et le mot `edge` par `edgedefault`.

Pour décrire les algorithmes à appliquer aux graphes, il faut utiliser une description similaire à celles utilisées pour les nœuds ou les arêtes :

```
<algorithmes>::=algorithm(type=<entier>,placement=<entier>,
                  nbsteps=<entier>,start=<entier>,end=<entier>,
                  dopt=<entier>,dmax=<entier>,
                  2d=<entier>,nodeedgerepuls=<entier>
                )
```

La valeur du champ `type` doit être 0 pour l'algorithme pour graphes non orientés, 1 pour l'algorithme pour graphes orientés, et 2 pour l'algorithme squelette du graphe. La valeur du champ `placement` doit être 0 pour l'algorithme incrémental de placement dans les graphes orientés, et 1 pour la méthode *greedy*. La fenêtre contenant les options des algorithmes de tracé est mise à jour automatiquement après la lecture des données concernant l'algorithme à utiliser.

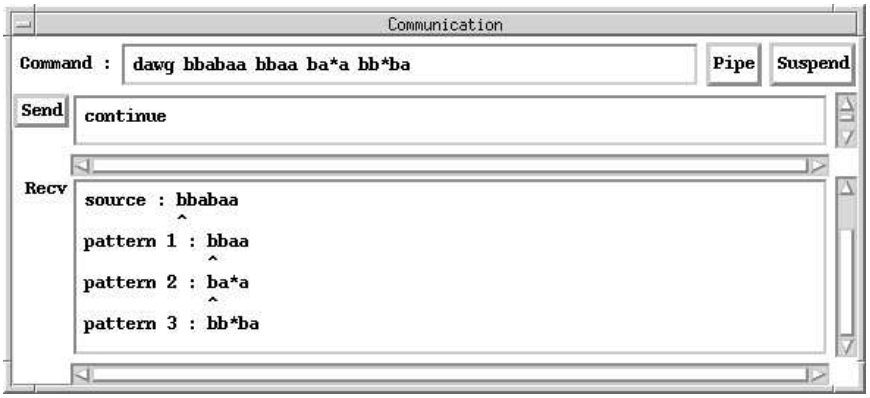
Voici un exemple complet de graphe représentant un cube formé de sept nœuds en forme de cubes bleus et d'un nœud en forme de sphère rouge, avec des arêtes jaunes. Un des nœuds est étiqueté par le mot `un nœud`, et une des arêtes par `arête`. Le nœud rouge contient une information qui apparaît quand on sélectionne le nœud dans la fenêtre de visualisation, dans une fenêtre dédiée.


```
IMPORT
```

```
nodedefault(color=[0,0,255],shape=Cube)
node(id=1,shape=Sphere,color=[255,0,0],
     info='Quelques informations
sur le nœud rouge
peuvent être affichées dans
cette fenêtre')
node(id=2)
node(id=3)
node(id=4,label='un nœud')
node(id=5)
node(id=6)
node(id=7)
node(id=8)
edgedefault(color=[255,255,0])
edge(source=1,target=2)
edge(source=2,target=3)
edge(source=3,target=4)
edge(source=4,target=1)
edge(source=5,target=6)
edge(source=6,target=7,label='arête')
edge(source=7,target=8)
edge(source=8,target=5)
edge(source=1,target=5)
edge(source=2,target=6)
edge(source=3,target=7)
edge(source=4,target=8)
.
```

1.4 Communications

1.4.1 Fenêtre de communication



La fenêtre de communications permet de relier le programme `Padnon` à un autre programme. Le champ `Command` permet d'entrer le nom du programme avec ses paramètres. Le programme s'exécute après appui sur le bouton `Send`. Il est possible de suspendre puis de reprendre l'exécution avec le bouton `Suspend`. Les données parvenant sur la sortie standard du programme relié sont affichés dans la zone `Recv` située en bas de la fenêtre de communication, sauf pour certaines données particulières comme expliqué plus loin. Il est possible d'envoyer des données sur l'entrée standard du programme relié, à l'aide de la zone `Send`, puis en appuyant sur le bouton `Send`.

Deux programmes particuliers peuvent être reliés à `Padnon`. Le programme `catlayout` prend en argument un nom de fichier contenant la description d'un graphe. Le graphe est chargé puis dessiné automatiquement dans `Padnon`. Le second programme, appelé `catanim`, a le même usage, mais les différentes étapes du tracé du graphe sont affichées sous forme d'animation.

1.4.2 Modifications dynamiques et tracé

Certains mots clefs, quand ils sont lus par `Padnon`, déclenchent des actions particulières portant sur le graphe. Il convient d'utiliser :

- `IMPORT` pour indiquer que les données suivantes correspondent à la description d'un graphe. La description du graphe se termine quand `Padnon` rencontre un point seul sur une ligne.
- `DELETENODE` pour indiquer que la ligne suivante contient une liste d'identificateurs de nœuds à supprimer, par exemple `[1,3,5]`.
- `DELETEEDGE` pour indiquer que la ligne suivante contient une liste d'identificateurs d'arêtes à supprimer.
- `DELETEALL` pour indiquer que le graphe entier doit être supprimé.
- `LAYOUT` pour effectuer toutes les étapes de l'algorithme de tracé automatique.
- `STARTLAYOUT` pour effectuer la première étape de l'algorithme de tracé.
- `STEPLAYOUT` pour effectuer l'étape suivante de l'algorithme de tracé.
- `ACK` pour demander à ce que la donnée `ack` soit envoyée sur l'entrée standard du programme relié, ce qui permet de synchroniser les deux programmes. Dans le cas d'un programme en langage `C`, on écrira par exemple :

```
char tmp[4];
```

```
printf("ACK\n");fflush(stdout);
scanf("%s",tmp);
```

1.5 Réalisation

Le programme `Padnon` est écrit en `C++`. Il s'appuit sur la bibliothèque `Inventor` [7] pour la représentation en trois dimensions des graphes, et la bibliothèque `Motif` [6] pour l'interface utilisateur du programme. L'éxécutable `Padnon` est disponible par ftp anonyme à l'adresse `ftp://ftp.loria.fr/pub/loria/eureca/PADNON/padnon.tar.gz`.

2 Algorithmes de tracé

Les deux algorithmes de tracé, dans le cas de graphes orientés et non orientés, s'appuient sur un principe simple, inspiré des interactions entre des particules d'un système physique. Cette méthode est présentée initialement dans [3], puis étendue dans [4] et [1] : deux nœuds du graphe ont tendance à s'écarter l'un de l'autre, comme le feraient deux particules chargées positivement. Une arête du graphe tend à empêcher les nœuds reliés de s'écarter trop, comme si deux particules, de même charge, étaient reliées par un élastique. L'algorithme est itératif, et le déplacement des nœuds à chaque étape est borné. Par rapport aux méthodes classiques utilisant ces algorithmes, nous ajoutons une contrainte supplémentaire visant à faire s'écarter les nœuds des arêtes.

À chaque nœud a du graphe, nous associons les informations suivantes :

- le vecteur position du nœud, noté $position(a)$,
- le vecteur correspondant à la force exercée sur le nœud, noté $force(a)$, dont dépend le déplacement du nœud à chaque étape de l'algorithme,
- le vecteur $fixe(a)$, dont la i^{e} composante, à valeur booléenne, indique si la i^{e} composante du vecteur position du nœud peut être modifiée ou non au cours de l'exécution de l'algorithme.

À chaque arête e du graphe, nous associons les extrémités de l'arête notées $s(e)$ et $t(e)$. Dans le cas orienté, $s(e)$ correspond à l'origine de l'arête, $t(e)$ à sa destination.

Notation. Dans la suite, nous considérons que les vecteurs $position(a)$, $force(a)$ et $fixe(a)$ sont de dimension 3. Nous noterons $a.x$ la première composante du vecteur $position(a)$, $a.y$ et $a.z$ les deuxième et troisième. De même les composantes de $force(a)$ seront notées $a.dx$, $a.dy$ et $a.dz$, et celles de $fixe(a)$ notées $a.fx$, $a.fy$ et $a.fz$.

Une étape de calcul de l'algorithme consiste à initialiser à zéro les forces exercées sur les noeuds, calculer les forces de répulsion pour chaque couple de noeuds, puis les forces d'attraction entre les noeuds reliés par une arête, et enfin les forces de répulsion entre les nœuds et les arêtes. Le déplacement d'un nœud correspond simplement à la force exercée, dont la norme est bornée par la valeur $borne\_deplacement$. L'opération $masque(force(a), fixe(a))$ retourne un vecteur dont la i^{e} composante a pour valeur la i^{e} composante de $force(a)$, si la i^{e} composante de $fixe(a)$ vaut `faux`, et 0 si elle vaut `vrai`. Dans les algorithmes suivants, nous notons $||v||$ la norme euclidienne d'un vecteur v.

Algorithme (Étape de calcul).

```
procedure etape(borne_deplacement:entier)
   pour tout nœud a faire
      force(a):=0⃗;
   fin_pour;

   pour tout nœud a faire
      pour tout nœud b faire
         repulsion(a,b);
      fin_pour;
   fin_pour;

   pour toute arete e faire
      attraction(s(e),t(e));
   fin_pour;

   pour tout nœud a faire
      si ||force(a)|| > borne_deplacement alors
         force(a):=(force(a) borne_deplacement)/||force(a)||;
      fin_si;
      position(a):=position(a)+masque(force(a),fixe(a));
   fin_pour;

   pour tout nœud a faire
      pour toute arete e faire
         repulsion_nœud_arete(a,e);
      fin_pour;
   fin_pour;
fin_procedure;
```

En termes de complexité, comme nous le verrons par la suite, chaque calcul d'interaction nécessite un nombre fini d'opérations arithmétiques. Le coût d'une étape

de calcul est donc en $O(n^2) + O(m.n)$, où n correspond au nombre de nœuds, et m au nombre d'arêtes.

Si $d_{opt}(a, b)$ représente la distance idéale entre les centres des nœuds a et b reliés par une arête, alors on peut définir les forces d'attractions qui portent sur a et b de sorte que si deux noeuds a et b, reliés par une arête, se trouvent à distance $d_{opt}(a, b)$ l'un de l'autre, la somme des forces d'attraction et de répulsion exercée sur les nœuds est nulle. De plus, nous faisons en sorte que la force de répulsion entre deux nœuds diminue au fur et à mesure que les nœuds s'écartent. La valeur de d_{opt} peut être calculée en fonction des dimensions des nœuds.

2.1 Graphes non orientés

La force appliquée aux nœuds est calculée de la même façon pour chaque composante du vecteur :

Algorithme (Attraction des nœuds).

```
procedure attraction(a:nœud, b:nœud)
   d:=||ab||;
   f:=d²/d_opt(a,b);
   force(a):=force(a)+f · ab/d;
   force(b):=force(b)-f · ab/d;
fin_procedure;
```

Nous choisissons de ne pas tenir compte des forces de répulsion entre deux nœuds distants de plus d'une valeur d_{max}. Cela permet de tracer des graphes non connexes sans être obligé d'ajouter d'arête "invisible".

Algorithme (Répulsion des nœuds).

```
procedure repulsion(a:nœud, b:nœud)
   d:=||ab||;
   si d≤d_max alors
     f:=-d_opt(a,b)²/d;
     force(a):=force(a)+f · ab/d;
     force(b):=force(b)-f · ab/d;
   fin_si;
fin_procedure;
```

Pour faire en sorte qu'un nœud a s'écarte d'une arête e, nous procédons comme suit. Nous regardons si la projection orthogonale i du point où se situe le nœud

a appartient au segment formé par les extrémités de l'arête. Si c'est le cas, nous calculons les forces de répulsion entre un nœud imaginaire situé au point i et le nœud a. La force qui devrait s'appliquer au nœud i est ensuite appliquée aux nœuds extrémités de l'arête.

Algorithme (Interaction entre nœuds et arêtes).

```
procedure repulsion_nœud_arete(a:nœud, e:arete)
   s:=s(e);
   t:=t(e);
   α:=(sa⃗ · st⃗)/||st⃗||²;
   si 0 ≤ α≤1 alors
      soit i tel si⃗ = α · st⃗;
      d:=||ia⃗||;
      si d≤d_max alors
         f:=-d_opt(a,e)²/d;
         force(a):=force(a)+f · ia⃗/d;
         force(s):=force(s)-f · ia⃗/d;
         force(t):=force(t)-f · ia⃗/d;
      fin_si;
   fin_si;
fin_procedure;
```

2.2 Graphes orientés

Dans le cas des graphes orientés, nous distinguons l'une des composantes du vecteur position, la composante *verticale*, qui correspond à la valeur $a.z$ d'un nœud a. Le calcul des forces diffère selon les composantes considérées. Le placement des nœuds selon la composante verticale dépend de l'ordre dans lequel les nœuds et arêtes sont placés. En contre-partie, le dessin, lors de l'ajout d'un nœud ou d'une arête n'est modifié que légèrement. Nous introduisons de nouvelles données pour chaque nœud a du graphe :

- le *nœud d'attache* de a, noté $attache(a)$ est le noeud à partir duquel nous calculons la position de a,
- la *position d'attache* de a, notée $posattache(a)$, est la position relative de a par rapport à son nœud d'attache,
- l'*identificateur de composante*, noté $composante(a)$, est un entier qui est identique entre deux nœuds appartenant à la même composante connexe du graphe.

Les nœuds d'attache sont calculés au fur et à mesure que l'on ajoute des arêtes au graphe. Lors de l'ajout d'un nœud a, $attache(a)$ est initialisé à `nil`, et $composante(a)$ reçoit une valeur non encore utilisée. Lors de l'ajout d'une arête, on regarde tout d'abord si les deux nœuds reliés font partie de la même composante connexe. Si c'est le cas, on ne fait rien. Sinon, si l'une des extrémités de l'arête n'a pas de nœud d'attache, on lui attache l'autre extrémité. La position d'attache est déterminée en fonction de la valeur $\delta_{opt}(a,b)$, la distance verticale optimale entre les nœuds a et b. Les nœuds a et b font maintenant partie de la même composante connexe : on met donc à jour les valeurs des composantes des nœuds.

Algorithme (Ajout d'une arête).

```
procedure ajoute_arete(a:nœud, b:nœud)
   si composante(a)≠composante(b) alors
      si attache(b)=nil alors
         attache(b)=a;
         pos_attache(b)=δ_opt(a,b);
      sinon si attache(a)=nil alors
         attache(a)=b;
         pos_attache(a)=−δ_opt(a,b);
      fin_si;
      tmp:=composante(b);
      pour tout nœud n faire
        si composante(n) = tmp alors
           composante(n):=composante(a);
        fin_si;
      fin_pour;
   fin_si;
fin_procedure;
```

L'algorithme ci-dessus, en ne permettant de créer des liens *attache* qu'entre des nœuds appartenant, avant l'ajout de l'arête, à des composantes connexes différentes, garantit qu'il n'y a pas de cycles selon les liens *attache*. Le coût de l'algorithme est en $O(n)$, où n est le nombre de nœuds du graphe.

Dans le cas orienté, nous dirons que deux nœuds sont situés à un *même niveau* s'ils sont séparés par une distance verticale inférieure à un nombre δ. Pour deux nœuds situés au même niveau, nous appliquons les même procédures que dans le cas non-orienté, mais en ignorant la composante verticale. Si deux nœuds ne sont pas sur un même niveau, nous ignorons les forces de répulsion ; si ces deux nœuds sont reliés par une arête, nous essayons de faire en sorte que l'arête soit la plus verticale

possible. Dans les procédures qui suivent, nous notons $reduit(v)$ la restriction du vecteur v à ses deux premières composantes :

Algorithme (Attraction des nœuds).

```
procedure attraction(a:nœud, b:nœud)
   dz:=|b.z − a.z|;
   si dz > δ alors
      reduit(force(a)):=reduit(force(a)) + reduit(ab)/2;
      reduit(force(b)):=reduit(force(a)) − reduit(ab)/2;
   sinon
      d:=||reduit(ab)||;
      si d≤d_max alors
         f:=d²/d_opt(a,b);
         reduit(force(a)):=reduit(force(a)) + f · reduit(ab)/d;
         reduit(force(b)):=reduit(force(b)) − f · reduit(ab)/d;
      fin_si;
   fin_si;
fin_procedure;
```

Algorithme (Répulsion des nœuds).

```
procedure repulsion(a:nœud, b:nœud)
  dz:=|b.z − a.z|;
   si dz ≤ δ alors
      d:=||reduit(ab)||;
      f:=−d_opt(a,b)²/d;
      reduit(force(a)):=reduit(force(a)) + f · reduit(ab)/d;
      reduit(force(b)):=reduit(force(b)) − f · reduit(ab)/d;
   fin_si;
fin_procedure;
```

La phase d'écartement entre les nœuds et les arêtes se fait selon le même principe que dans le cas non-orienté. Cependant, on ne calcule plus la projection orthogonale sur la droite passant par l'arête, mais l'intersection entre le plan horizontal passant par le nœud et la droite passant par l'arête.

Algorithme (Répulsion des nœuds).

```
procedure repulsion_nœud_arete(a:nœud, e:arete)
   s:=s(e);
   t:=t(e);
```

```
   α:=(a.z−s.z)/(t.z−a.z);
   si 0 ≤ α≤1 alors
      soit i tel si⃗ = α · st⃗;
      d:=||ia⃗||;
      si d≤d_max alors
         f:=−d_opt(a,b)²/d;
         reduit(force(a)):=reduit(force(a))+f · reduit(ai⃗)/d;
         reduit(force(b)):=reduit(force(b))-f · reduit(ai⃗)/d;
      fin_si;
   fin_si;
fin_procedure;
```

En plus des attractions et des répulsions entre nœuds, il convient d'essayer, à chaque étape, de placer chaque nœud en fonction de son père d'attache.

Algorithme (Attachement des nœuds).

```
procedure attachement(a:nœud)
   si attache(a)≠nil alors
      b:=attache(a);
      a.fz:=a.fz + b.z + posattache(a) − a.z;
   fin_si;
fin_procedure;
```

3 Conclusion

Nous avons présenté un système permettant le tracé automatique de graphes, orientés et non orientés, en trois dimensions. L'algorithme de tracé présenté permet, après une petite modification du graphe, d'obtenir une représentation nouvelle tenant compte du tracé dont on disposait avant modification. Ceci permet de conserver la représentation mentale que l'on s'est faite du graphe au cours des étapes précédentes. Cette propriété rend le programme `Padnon` particulièrement bien adapté à la représentation dynamique des structures de données de programmes, et peut donc servir comme interface pour l'animation d'algorithmes.

Références

[1] T. Kamada ans S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.

[2] G. Di Battista, P. D. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography, 1993. Disponible par ftp anonyme `ftp://wilma.cs.brown.edu/pub/gdbiblio.ps.Z`.

[3] P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[4] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software-Practice and Experience*, 21(11):1129–1164, 1991.

[5] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM J. of Algebraic and Discrete Methods*, 4(3):312–316, 1983.

[6] D. Heller and M. Paula. *Motif programming manual : for OSF/Motif release 1.2.* O'Reilly and Associates, 1994.

[7] J. Wenecke. *The Inventor Mentor.* Addison Wesley, 1994.

Unit´e de recherche INRIA Lorraine, Technop^ole de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rh^one-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399